The Computational Complexity Column

by

Lance FORTNOW

NEC Research Institute
4 Independence Way, Princeton, NJ 08540, USA
fortnow@research.nj.nec.com

This summer I take over two positions from Eric Allender: Editor of this column and chair of the conference committee of the IEEE Conference on Computational Complexity. While these jobs have quite different responsibilities, I believe they share the same goals: To promote computational complexity and keep researchers informed of the latest happenings in the area. I thank Eric for his excellent work on both this column and the conference. His shoes will be hard to fill.

In my first column I give a survey on "Diagonalization" based on a talk given at the recent DIMACS workshop. I hope you find it interesting.

This column should not be a one-way street. Please feel free to contact me if you have comments about this column (or the conference) particularly if you have ideas for future column topics.

Diagonalization[1]

Lance Fortnow

Abstract

We give a modern historical and philosophical discussion of diagonalization as a tool to prove lower bounds in computational complexity. We will give several examples and discuss four possible approaches to use diagonalization for separating logarithmic-space from nondeterministic polynomial-time.

1 Introduction

The greatest embarrassment in computational complexity theory comes from our inability to achieve significant complexity class separations. In recent years we have seen many interesting results come from an old technique—diagonalization. Deceptively simple, diagonalization, combined with techniques for collapsing classes, can yield quite interesting lower bounds on computation.

In 1874, Cantor [Can74] first used diagonalization for showing the set of reals is not countable. The proof worked by assuming an enumeration of the reals and designing a set that one-by-one is different from every set in the enumeration. Drawn as a table this process considers the diagonal set and reverses it. Thus the term diagonalization.

Diagonalization was first used in computability theory in the 1930's by Turing [Tur36] to show that there existed computably enumerable problems that were not computable. The seminal paper in computational complexity [HS65] used diagonalization to give time and space hierarchies.

[1] Based on an invited talk given at the DIMACS Workshop on Computational Intractability on April 13, 2000. The slides of that presentation can be found at http://www.neci.nj.nec.com/homepages/fortnow/talks.

Diagonalization works! In Section 2.1 we describe Allender's diagonalization proof [All99] that the permanent is not computable by uniform constant depth threshold circuits. Without diagonalization, can we even show that the halting problem does not have such circuits?

Razborov and Rudich [RR97] has developed the concept of "natural proofs" that capture the known techniques for proving lower bounds in nonuniform circuit classes. They show that under reasonable assumptions, these proofs cannot give us strong lower bounds for various interesting circuit problems. Since diagonalization works against uniform models of computation, these issues do not apply.

The techniques for diagonalization remain relatively simple. To prove a separation result, one assumes a collapse and derives enough consequences until we violate a well-known time-hierarchy theorem. Most of the interesting diagonalization proofs do not rely on hard combinatorics.

Diagonalization does have its limitations. Most of the results we have by diagonalization are still far weaker than we would hope. Diagonalization also gives even weaker results against probabilistic and nonuniform computation.

Baker, Gill and Solovay [BGS75] develop the notion of "relativization". They created a relativized world A where $\mathbf{P}^A = \mathbf{NP}^A$. Since all of the known diagonalization proofs of the time relativize, this gave a good argument that diagonalization alone would not separate $\mathbf{P}$ from $\mathbf{NP}$. One can get some nonrelativizing separation results. We give one such example in Section 2.6. However these results require nonrelativizing collapses of which there are few.

We will give several results showing how diagonalization has and continues to play an important role in computational complexity theory. We also argue that diagonalization may even help us separate classes like $\mathbf{NP}$ from $\mathbf{L}$—we give four approaches towards this goal.

2 Diagonalization Proofs

In this section we give several examples about diagonalization results to give a taste and history of the technique.

2.1 Permanent is not in uniform $\mathbf{TC}_0$

A wonderful example of the diagonalization technique is the result showing that the permanent cannot be computed by uniform constant depth threshold circuits. This result was proven by Allender [All99] building on work of Caussinus, McKenzie, Thérien and Vollmer [CMTV98] and Allender and Gore [AG94]. We sketch the proof in this section.

Consider *threshold* machines: These are like alternating machines except that instead of asking existential and universal questions they ask "Do a majority of my computation paths accept?" A k-threshold machine can have k threshold questions on any path. Polynomial-time unbounded-threshold machines give us $\mathbf{PSPACE}$. Polynomial-time constant-threshold machines characterize the counting hierarchy. Logarithmic-time constant-threshold machines with random-access to the input characterize uniform $\mathbf{TC}_0$.

Suppose the permanent is in uniform $\mathbf{TC}_0$ and therefore in $\mathbf{P}$. Thus we can count computation paths in polynomial-time [Val79] so the entire counting hierarchy collapses to $\mathbf{P}$. The proofs that the permanent is $\#\mathbf{P}$-complete [Val79, Zan92] show that the permanent is in fact complete under reductions computable by constant depth circuits. Since the permanent is in $\mathbf{TC}_0$ the counting hierarchy now collapses to $\mathbf{TC}_0$.

By straightforward diagonalization one can get that for any fixed k there exists a language accepted by a polynomial-time k-threshold machine not accepted by any logarithmic-time k-threshold machine. We have not yet reached a contradiction since it is possible that a polynomial-time k-threshold machine is accepted by some logarithmic-time $k+1$-threshold machine.

Now **SAT** is accepted by a log-time k-threshold machine for some fixed k. All of **NP**, and thus the counting hierarchy is reducible to **SAT** via simple projections. All of the polynomial-time constant threshold machines can be simulated by a logarithmic-time k-threshold machine giving us the contradiction.

This proof is a great example of the diagonalization method: We want to prove a separation. First we assume the collapse. Then we get other collapses and keep on collapsing until we can apply a straightforward diagonalization.

2.2 Time and Space Hierarchies

The first uses of diagonalization in complexity theory came in the very first papers. The main results of the seminal paper in complexity theory by Hartmanis and Stearns [HS65] gave deterministic time and space hierarchy results using straightforward diagonalization.

Nondeterministic hierarchy results are not so straightforward because of the need to do the opposite in the diagonalization step. Ibarra [Iba72] showed how to get good bounds for the nondeterministic space hierarchy by making many collapses and then applying Savitch's Theorem [Sav70] to get the determinism needed for diagonalization. Immerman [Imm88] and Szelepcsényi [Sze88]'s results that nondeterministic space is closed under complement removed this problem and gave tight hierarchies.

We do not believe nondeterministic time is closed under complement and we also do not have any deterministic simulation nearly as nice as Savitch's theorem for space. However, using a large number of collapses we can achieve quite a tight hierarchy for nondeterministic time.

Cook [Coo73] first showed that $\mathbf{NTIME}(n^r) \subsetneq \mathbf{NTIME}(n^s)$ if $1 \leq r < s$. Seiferas, Fischer and Meyer [SFM78] give a significantly stronger version.

Theorem 2.1 (Seiferas-Fischer-Meyer) *For any time-constructible functions s and t such that $s(n+1) = o(t(n))$, there exists a language accepted in nondeterministic time $t(n)$ but not accepted in nondeterministic time $s(n)$.*

We sketch a simple proof of Theorem 2.1 due to Žàk [Žàk83].

Sketch of Proof: Let $M_1, \ldots$ be an enumeration of nondeterministic Turing machines. We define a nondeterministic machine M that acts as follows on input $w = 1^i01^m01^k$: If $k < m^{t(m)}$ then simulate M_i on input $1^i01^m01^{k+1}$ for $t(|w|)$ steps. If $k = m^{t(m)}$ then accept if 1^i01^m0 rejects which we can do quickly as a function of the current input size.

This machine uses time $t(n)$ so by assumption can be simulated in time $s(n)$ by some machine M_i. Since $s(n+1) = o(t(n))$ we have for sufficiently large m,

$$1^i01^m0 \in L(M) \Leftrightarrow 1^i01^m01 \in L(M) \Leftrightarrow \cdots \Leftrightarrow 1^i01^m01^{m^{t(m)}} \in L(M) \Leftrightarrow 1^i01^m0 \notin L(M)$$

a contradiction. $\square$

2.3 Delayed Diagonalization

In 1975, Ladner [Lad75] showed that if $\mathbf{P} \neq \mathbf{NP}$ there must an incomplete set in $\mathbf{NP} - \mathbf{P}$. His proof creates a set that is sometimes **SAT** to keep it out of **P** and sometimes the empty set to keep it incomplete. The tricky part is to keep the set in **NP**. We need to be patient and wait until we actually see a diagonalization occur.

Theorem 2.2 (Ladner) *If* $\mathbf{P} \neq \mathbf{NP}$ *then there exists a set* A *such that*

1. $A \in \mathbf{NP}$,
2. $A \notin \mathbf{P}$, *and*
3. A *is not* $\mathbf{NP}$*-complete.*

Proof: We will create a polynomial-time computable function $f : 1^* \to \mathbb{N}$. We define our set A as

$$A = \{\phi \mid \phi \in \mathbf{SAT} \text{ and } f(|\phi|) \text{ is even}\}.$$

Clearly A is in $\mathbf{NP}$. We will use f to indicate our current stage. When f is equal to $2i$ then we will try to prevent A from being accepted by the ith polynomial-time Turing machine. When f is equal to $2i+1$ we will prevent $\mathbf{SAT}$ from reducing to A via the ith polynomial-time computable reduction. To keep f polynomial-time computable we will need to wait not only for the diagonalization to occur but for there to be enough time for us to see it. Thus the notion of "delayed diagonalization."

Let $M_1, \ldots$ be an enumeration of the polynomial-time computable Turing machines. Let $g_1, \ldots$ be an enumeration of polynomial-time computable functions.

Initially let $f(0) = 2$. We will start in stage $n = 1$.

STAGE n:

Let $j = f(n-1)$. If j is even we will work against $A = L(M_{\frac{j}{2}})$. If j is odd we will work against $g_{\lfloor \frac{j}{2} \rfloor}$ reducing $\mathbf{SAT}$ to A.

CASE $j = 2k$:

See if there exists a formula ϕ such that $|\phi| \leq \log n$ and ϕ is in the symmetric difference of $L(M_k)$ and A as defined so far. If so let $f(n) = j + 1$ otherwise let $f(n) = j$.

CASE $j = 2k + 1$:

See if there exists a formula ϕ such that $|\phi| \leq \log n$ and either

1. $\phi \in \mathbf{SAT}$ and $g_k(\phi) \notin A$, or
2. $\phi \notin \mathbf{SAT}$ and $g_k(\phi) \in A$.

If so then let $f(n) = j + 1$ otherwise let $f(n) = j$.

If $f(n)$ goes to infinity then we have fulfilled the conditions for A. If $f(n)$ reaches a limit of $2k$ then A will be equal to $L(M_k)$ and also finitely different from $\mathbf{SAT}$ violating the $\mathbf{P} \neq \mathbf{NP}$ assumption. Likewise if $f(n)$ reaches a limit of $2k + 1$ then g_k will reduce $\mathbf{SAT}$ to A but A will be finite again violating the $\mathbf{P} \neq \mathbf{NP}$ assumption. $\square$

2.4 Every separation is diagonalization?

Partly in response to the relativization work of Baker, Gill and Solovay [BGS75], Kozen [Koz80] took a different tact. He argued that any proof of say $\mathbf{P} \neq \mathbf{NP}$ would be a proof by diagonalization.

Let $M_1, \ldots$ be an enumeration of the polynomial-time computable machines. Let h be a function mapping Σ^* to the natural numbers. We define $\mathbf{diag}_h$ as

$$\mathbf{diag}_h = \{x \mid M_{h(x)}(x) \text{ rejects}\}.$$

Kozen notes that either of the following two conditions guarantee that $\mathbf{diag}_h$ is not in $\mathbf{P}$.

1. For all L in $\mathbf{P}$ there is some x such that $L = L(M_{h(x)})$.
2. For all L in $\mathbf{P}$ there is L' such that L' and L differ in finitely many places and for infinitely many x, $L' = L(M_{h(x)})$.

Theorem 2.3 (Kozen) *For any computable set B not in $\mathbf{P}$ there exists a computable h such that $B = \mathbf{diag}_h$. Moreover, h can be chosen to meet both of the conditions above.*

In particular if $\mathbf{P} \neq \mathbf{NP}$ then we can take $B = \mathbf{SAT}$ in Theorem 2.3. Any proof that $\mathbf{P} \neq \mathbf{NP}$ would imply a proof of the existence of an h fulfilling the conditions above such that $\mathbf{SAT} = \mathbf{diag}_h$.

I believe this result says more about the difficulty of exactly formalizing the notion of a "diagonalization proof" than of actually arguing the diagonalization technique is the only technique we have for class separation.

2.5 Separations against nonuniform classes

Diagonalizing against nonuniform classes appears quite difficult. One could use some input to diagonalize against a particular circuit. Unfortunately we usually have more circuits than inputs.

Kannan [Kan82] gives an interesting strategy for showing that some classes do not have small circuits.

Theorem 2.4 (Kannan)

1. *For any fixed k, there is a language in $\Sigma_2^p \cap \Pi_2^p$ not computable in n^k-size circuits.*
2. *There is a language in $\Sigma_4^{\mathbf{E}} \cap \Pi_4^{\mathbf{E}}$ not computable by $2^{o(n)}$-size circuits.*
3. *There is a language in $\Sigma_2^{\mathbf{E}} \cap \Pi_2^{\mathbf{E}}$ not computable by polynomial-size circuits.*

The class $\Sigma_k^{\mathbf{E}}$ represents the $2^{O(n)}$ version of Σ_k^p.

Proof: We will give the proof for the first. The other two are similar.

Fix k. Consider the set of strings L consisting of x accepted by the lexicographically least circuit of size n^{k+1} that is different from all circuits of size n^k. Simple counting arguments show that such circuits must exist. This expression can be formulated in Σ_4^p.

To get a separation at the second level of the hierarchy we use a nonconstructive argument. If $\mathbf{SAT}$ does not have n^k size circuits than the result follows. Otherwise by Karp and Lipton [KL80] the entire polynomial-time hierarchy and thus L is contained in $\Sigma_2^p \cap \Pi_2^p$. $\Box$

Is this proof a diagonalization argument or really a simple combinatorial argument? It is not clear and an informal survey of fellow complexity theorists gave a mixed response.

2.6 Nonrelativizing Separations

Buhrman, Fortnow and Thierauf [BFT98] give the first separation result that does not relativize. Consider the class $\mathbf{MA_{EXP}}$ that consists of languages proven by an interactive proof system where the prover sends a single message to a probabilistic exponential-time verifier.

Theorem 2.5 (Buhrman-Fortnow-Thierauf)

1. *There exists a language in* $\mathbf{MA_{EXP}}$ *that does not have polynomial-size circuits.*

2. *There exists a relativized world where every language in* $\mathbf{MA_{EXP}}$ *has polynomial-size circuits.*

Proof of Theorem 2.5(1): Assume that $\mathbf{MA_{EXP}}$ has polynomial-size circuits. This implies that $\mathbf{EXP}$ has polynomial-size circuits and thus that $\mathbf{EXP} = \mathbf{MA}$ [BFL91]. We then have $\Sigma_2^p \subseteq \mathbf{MA}$ and by translation that $\Sigma_2^{\mathbf{EXP}} \subseteq \mathbf{MA_{EXP}}$. This contradicts Kannan's result [Kan82] that $\Sigma_2^{\mathbf{EXP}}$ does not have polynomial-size circuits. $\square$

The proof does not relativize because it relies on the result of Babai, Fortnow and Lund [BFL91] that $\mathbf{EXP}$ has polynomial-size circuits implies $\mathbf{EXP} = \mathbf{MA}$ which follows from their nonrelativizing proof of $\mathbf{MIP} = \mathbf{NEXP}$.

This proof shows that one can get nonrelativizing diagonalization arguments by using nonrelativizing collapses.

3 Approaches to separating L from NP

While the $\mathbf{P} \neq \mathbf{NP}$ question remains quite formidable, the $\mathbf{L} \neq \mathbf{NP}$ question seem much more tractable. We have no reason to think this question is difficult. The lack of good relativization models for space means we have no meaningful oracle model where $\mathbf{L}$ and $\mathbf{NP}$ collapse. Also since $\mathbf{L}$ is a uniform class, the Razborov-Rudich [RR97] limitations do not apply.

In this section we give four different approaches to attack this problem.

3.1 Autoreducibility

Trakhtenbrot [Tra70a] first looked at autoreducibility in the computability setting as a measure of redundancy in a set. Buhrman, Fortnow, van Melkebeek and Torenvliet [BFvMT00] showed that in the complexity setting autoreducibility may help separate complexity classes.

A set A is autoreducible if there exists an oracle polynomial-time Turing machine M such that $L(M^A) = A$ with the restriction that for all x, $M^A(x)$ does not query whether x is in A. Buhrman, Fortnow, van Melkebeek and Torenvliet show a relationship between complete sets and autoreducibility.

Theorem 3.1 (Buhrman-Fortnow-van Melkebeek-Torenvliet)

1. *If every Turing-complete set for* $\mathbf{EXPSPACE}$ *is autoreducible then* $\mathbf{NL} \neq \mathbf{NP}$.

2. *If every nonadaptively-Turing-complete set for* $\mathbf{PSPACE}$ *is nonadaptively autoreducible then* $\mathbf{NL} \neq \mathbf{NP}$.

Assuming $\mathbf{NL} = \mathbf{NP}$ Buhrman, Fortnow, van Melkebeek and Torenvliet create a series of constructions to get an A such that

1. A is in **EXPSPACE**,

2. A is Turing-hard for **EXPSPACE** and

3. A "diagonalizes" against all possible autoreductions.

They also give autoreductions for the **EXP**-complete sets and complete sets for some classes in the exponential-time hierarchy. These autoreductions use game characterizations of classes creating a contest between a player trying to show a string x is in a set A and a player trying to show that x is not in A. Earlier Beigel and Feigenbaum [BF92] used a different technique to show that all of the Turing-complete sets for **PSPACE** are autoreducible.

3.2 Intersecting Finite Automata

Karakostas, Lipton and Viglas [KLV00] give an interesting approach to the $\mathbf{L} \neq \mathbf{NP}$ problem by looking at the complexity of determining whether a collection of finite automata accept a common string.

Given a finite automaton of s states one can determine whether the machine accepts any strings at all in $O(s)$ time by using depth-first search to determine if there exists a path from the initial state to an accept state. If we are given k such automata, we can first create the intersecting automaton by using cross products and then apply depth-first search to this automata in $O(s^k)$ time. Karakostas, Lipton and Vigas show that even small improvements in this running time would yield complexity class separations.

Theorem 3.2 (Karakostas-Lipton-Viglas) *Suppose we are given k finite automata with s states and one additional automaton with t states. Let L be the intersection of the languages accepted by these automata.*

1. *If we can determine whether L is not empty in $s^{o(k)}t$ time then $\mathbf{L} \neq \mathbf{P}$.*

2. *If we can determine whether L is not empty with $s^{o(k)}t$-size circuits then $\mathbf{L} \neq \mathbf{NP}$.*

The proof works by assigning finite automata $F_1, \ldots, F_k$ to different regions of the work tape. Each F_i is responsible for checking the computation when the head is in its region. Each F_i has to keep track of its region of the tape and the current tape head location. An additional automaton G keeps track of the input tape. With appropriate choices of s for the sizes of the F_i and t for the size of G, if we can determine that L is not empty in $s^{o(k)}t$ time then we have $\mathbf{L} \subseteq \mathbf{DTIME}(n^{1+\epsilon}) \subsetneq \mathbf{P}$.

A similar proof shows that if we can determine whether L is not empty with $s^{o(k)}t$-size circuits then $\mathbf{L}$ has $n^{1+\epsilon}$ size circuits. If $\mathbf{L} = \mathbf{NP}$ then $\mathbf{L} = \Sigma_2^p$ and Σ_2^p cannot have n^k size circuits for any fixed k [Kan82].

We may have trouble applying Theorem 3.2 directly to separate $\mathbf{L}$ from $\mathbf{NP}$ because determining whether L is not empty may just be a difficult problem. However, to separate $\mathbf{L}$ from $\mathbf{NP}$, we need only show a quick algorithm for checking that L is not empty under the assumption that $\mathbf{L} = \mathbf{NP}$. There we may have more hope.

3.3 Hardness versus Randomness

Impagliazzo and Wigderson [IW97] show how to completely derandomize **BPP** using a strong hardness assumption. Trying to show this or even stronger assumptions false can lead to complexity separations.

Theorem 3.3 (Impagliazzo-Wigderson) *If there exist languages in* $\mathbf{E}$ *that cannot be computed by* $2^{o(n)}$*-size circuits then* $\mathbf{P} = \mathbf{BPP}$*.*

This is a wonderful derandomization result—but that is the topic of another survey. Instead let us focus on the antecedent. The antecedent seems awfully strong—It is impossible to have a very large amount of advice to give a small improvement on time. However, proving it false would imply $\mathbf{P} \neq \mathbf{NP}$.

Theorem 3.4 *If* $\mathbf{P} = \mathbf{NP}$ *then there exist languages in* $\mathbf{E}$ *that cannot be computed by* $2^{o(n)}$*-size circuits.*

Proof: If $\mathbf{P} = \mathbf{NP}$ then $\mathbf{P} = \Sigma_4^p$ and by translation $\mathbf{E} = \Sigma_4^{\mathbf{E}}$. However by Kannan [Kan82], $\Sigma_4^{\mathbf{E}}$ has languages that require $2^{\Omega(n)}$-size circuits. □

A similar argument shows that if linear space has small circuits we can get weaker separations.

Theorem 3.5 *If* $\mathbf{L} = \mathbf{NP}$ *then there exist languages in* $\mathbf{DSPACE}(n)$ *that do not have* $2^{o(n)}$*-size circuits.*

It remains open whether even $\mathbf{SAT} \in \mathbf{NTIME}(n) \subseteq \mathbf{DSPACE}(n)$ can be computed by $2^{o(n)}$-size circuits. However, if $\mathbf{SAT}$ does not have small circuits then we already know $\mathbf{L} \neq \mathbf{NP}$. This leads to the following approach to separating those classes.

Theorem 3.6 *If every language in* $\mathbf{DSPACE}(n)$ *has* $2^{o(n)}$*-size circuits assuming that* $\mathbf{SAT}$ *can be computed in polynomial-size circuits then* $\mathbf{L} \neq \mathbf{NP}$*.*

One can think of $\mathbf{DSPACE}(n)$ as linear alternating time. We want to simulate linear alternating time with slightly subexponential circuits on the assumption that we can do one layer of alternation in polynomial-size circuits.

3.4 Alternation, Time and Space

A recent approach looks at using collapses on machines of small alternations. This approach has led to interesting time-space tradeoffs for satisfiability. Kannan [Kan84] had looked at similar techniques in 1984 followed by more recent work by Fortnow [For00], Lipton and Viglas [LV99], Tourlakis [Tou00] and Fortnow and van Melkebeek [FvM00].

We give an easy example showing time-space trade-offs on Σ_2^n time.

Theorem 3.7 (Fortnow-van Melkebeek) *There exists a language in* Σ_2^n *that cannot be computed by any deterministic random-access Turing-machine using* $n^{1.99}$ *time and* $O(\log n)$ *space.*

Proof Sketch: Suppose the theorem is false. By translation we have every language L in $\Sigma_2^{n^2}$ accepted by a deterministic Turing-machine M using $n^{3.98}$ time and $O(\log n)$ space.

We will simulate M by a $\Sigma_2^{n^{1.99}\log n}$ machine violating the Σ_2-time hierarchy which is proven similarly to Theorem 2.1. Nondeterministically guess the configurations of M at the time step $in^{1.99}$ for $0 \leq i \leq n^{1.99}$. Universally pick an $i < n^{1.99}$ and deterministically check that the configuration at time $in^{1.99}$ can go to configuration $(i+1)n^{1.99}$. □

Similar techniques show that Σ_k^n requires nearly n^k deterministic time for small space-bounded machines. If one could show that for some fixed k, Σ_k^n requires n^j time for all j then we have separated $\mathbf{L}$ from $\mathbf{NP}$.

We can also use these ideas to get time-space tradeoffs for satisfiability. Fortnow and van Melkebeek building on the earlier papers show that satisfiability cannot be solved in n^a time and $n^{o(1)}$ space for random-access Turing machines for any a less than the golden ration, about 1.618.

4 Conclusions

Diagonalization, once given up for dead, has returned still giving us new and interesting lower bounds. While the actual diagonalization step still remains easy, we have new tools and techniques for collapsing classes. As we have seen in this survey, better collapses lead to better separations.

Acknoweledgments

I wish to thank the organizers of the DIMACS workshop, Paul Beame, Steve Rudich and Andrew Yao, for inviting me to give the talk. I also thank the many participants of the workshop whom I had comments and discussions relating to diagonalization. I also thank Bill Gasarch and Dieter van Melkebeek for many helpful comments on this manuscript.

References

[AG94] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23:1026–1049, 1994.

[All99] E. Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 1999(7), August 1999.

[BF92] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2(1):1–17, 1992.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing separations. In *Proceedings of the 13th IEEE Conference on Computational Complexity*, pages 8–12. IEEE, New York, 1998.

[BFvMT00] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Using autoreducibility to separate complexity classes. *SIAM Journal on Computing*, 29(5):1497–1520, 2000.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P = NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[Can74] G. Cantor. Ueber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. *Crelle's Journal*, 77:258–262, 1874.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57(2):200–212, October 1998.

[Coo73] S. Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, August 1973.

[For00] L. Fortnow. Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences*, 60(2):337–353, April 2000.

[FvM00] L. Fortnow and D. van Melkebeek. Time-space tradeoffs for nondeterministic computation. In *Proceedings of the 15th IEEE Conference on Computational Complexity*. IEEE Computer Society, Los Alamitos, 2000. To appear.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[Iba72] O. Ibarra. A note concerning nondeterministic tape complexities. *Journal of the ACM*, 19(4):608–612, 1972.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[IW97] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on the Theory of Computing*, pages 220–229. ACM, New York, 1997.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55:40–56, 1982.

[Kan84] R. Kannan. Towards separating nondeterminism from determinism. *Mathematical Systems Theory*, 17(1):29–45, April 1984.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on the Theory of Computing*, pages 302–309. ACM, New York, 1980.

[KLV00] G. Karakostas, R. Lipton, and A. Viglas. On the complexity of intersecting finite state automata. In *Proceedings of the 15th IEEE Conference on Computational Complexity*. IEEE Computer Society, Los Alamitos, 2000. To appear.

[Koz80] D. Kozen. Indexings of subrecursive classes. *Theoretical Computer Science*, 11:277–301, 1980.

[Lad75] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.

[LV99] R. Lipton and A. Viglas. On the complexity of SAT. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 459–464. IEEE, New York, 1999.

[RR97] A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, August 1997.

[Sav70] W. Savitch. Relationship between nondeterministic and deterministic tape classes. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[SFM78] J. Seiferas, M. Fischer, and A. Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, 1978.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tou00] I. Tourlakis. Time-space lower bounds for SAT on uniform and non-uniform machines. In *Proceedings of the 15th IEEE Conference on Computational Complexity*. IEEE Computer Society, Los Alamitos, 2000. To appear.

[Tra70a] B. Trakhtenbrot. On autoreducibility. *Doklady Akademii Nauk SSSR*, 192:1224–1227, 1970. In Russian. English Translation in [Tra70b].

[Tra70b] B. Trakhtenbrot. On autoreducibility. *Soviet Mathematics–Doklady*, 11:814–817, 1970.

[Tur36] A. Turing. On computable numbers, with an application to the Etscheidungs problem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Žàk83] S. Žàk. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.

[Zan92] V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2:77–82, 1992.